MR-Servo4433
User Manual

CONTENTS

PART1: MR-Servo4433

1. Introduction

MR-Servo4433 is a small pre-assembled R/C(Radio Control) servomotor controller, which has 16 R/C servo connectable I/O pins. The MR-Servo4433 can control up to 8 R/C servos at the same time. The MR-Servo4433 uses an AT90S4433(Atmel AVR series) CPU chip as a controller. The AT90S4433 has a 4K bytes In-System Programmable Flash memory, 128 bytes SRAM, 256 bytes EEPROM and many other peripherals. The user can download a program to the board without a ROM Writer using the ISP function. A free C-compiler (Microrobot AVR GCC) is provided.

2. Features

- AT90S4433 (Atmel AVR series, 8MHz(8 MIPS))
- 4Kbyte ISP flash, 128 bytes SRAM, 256 bytes EEPROM, 2 Timers, ADC 6ch, UART
- Four R/C servomotors
- 16 I/O port pins
- Controls up to 8 R/C servomotors at the same time
- C source code
- Free Windows C compiler(Microrobot AVR GCC)
- ISP downloader(Optional)
- On board piezo Buzzer

3. Control

The board has sixteen I/O port pins and can control 8 servomotors at the same time. The AT90S4433 CPU has two internal counters. The board generates up to eight periodic pulses using the timers. The periodic pulses control R/C servomotors.

PART 2 : BOARD

1. Placement Diagram(Silkscreen)

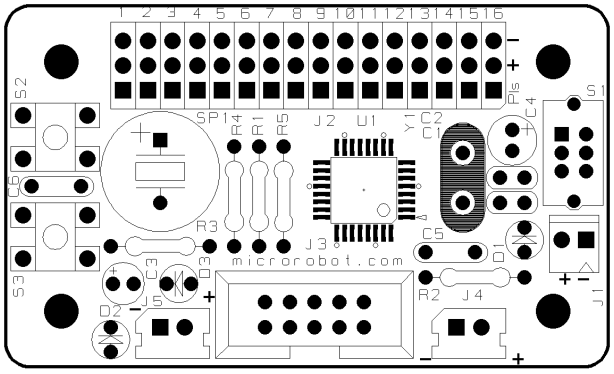
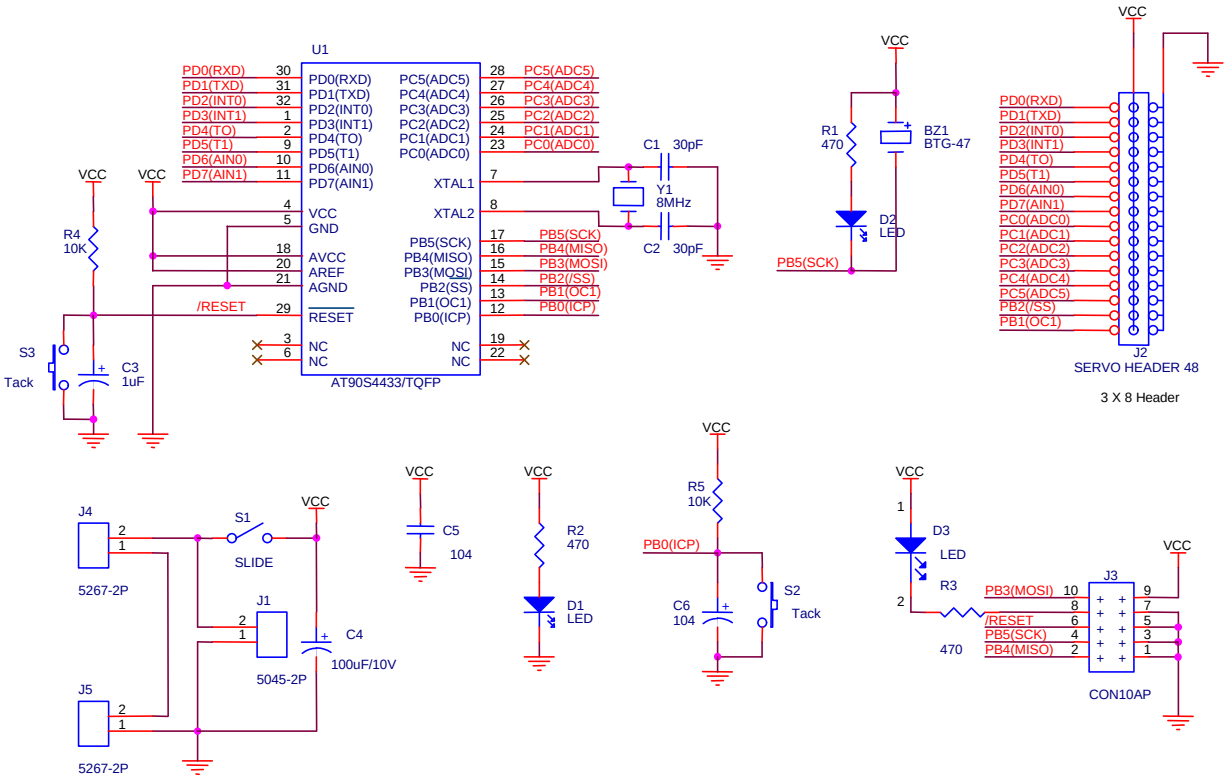


Fig 1.1 AT90S4433 Servomotor control board silkscreen.

2. Circuit Diagram



3. Parts List

NO	Reference	Parts name	Value	Qty.	Remark
1	C1, C2	Capacitor	30pF	2	Ceramic Condenser
2	C3	”	1uF	1	Electrolytic Condenser
3	C4	”	100uF/10V	1	Electrolytic Condenser
4	C5, C6	”	104(0.1uF)	2	Monolithic Condenser
5	D1, D2, D3	LED	RED 3ø	3	
6	J1	Connector	5045	1	5V Power Part
7	J2	”	HEADER PIN(Male)	1	SERVO HEADER 48PIN
8	J3	”	CON10AP	1	HIF3F/10PIN
9	J4, J5	”	5267	2	Battery Power Part
10	R1, R2, R3	Resistor	470Ω	2	
11	R4, R5	”	10K	2	
12	SP1	BUZZER	BTG-47	1	PIEZO
13	S1	S/W	SLIDE S/W	1	
14	S2, S3	”	Tack S/W	2	
15	U1	MCU	AT90S4433/TQFP	1	AVR Microcontroller
16	Y1	X-TAL	8MHz	1	ATS type
17		Printed Circuit Board(PCB)		1	Main PCB
18		Battery Holder & Power Connector	5051-2P	1	AA size * 4
19		Pin head Screw		4	3ø
20		Nut		12	3ø
21		Flat head Screw		4	3ø
22		Downloading Adapter		1	Option
23		Ribbon Cable		1	Option(1m)

Fig 2.1 Downloading Adapter

Fig 2.2 Ribbon cable

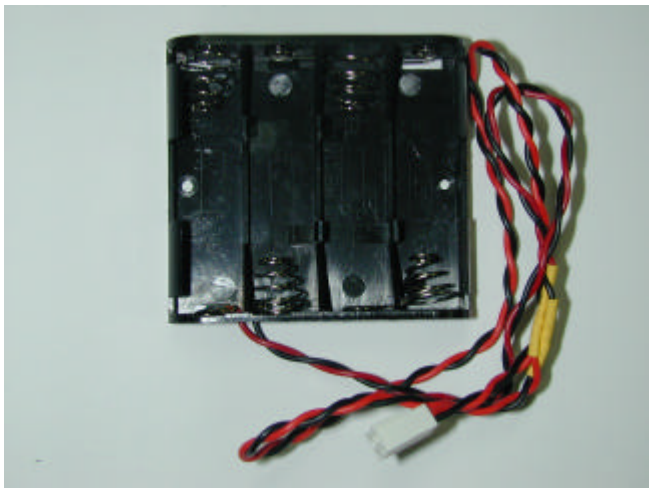


Fig 2.3 Battery Holder & Power Connector

PART 3 : Software Tools

1. AVR Development Program Installation

AVR Development Tools

There are many different kinds of development tools for AVR microcontrollers. Atmel, the AVR CPU manufacturer, provides some AVR development tools free. Microrobot Co. Ltd. also provides a free Windows C-compiler.

Wavrasm : AVR assembler, Atmel.
AVR Studio : AVR Emulator/Simulator, Atmel.
AVR ISP : ISP downloading program, Atmel.
Microrobot AVR GCC : C-compiler, Microrobot.

System requirements for AVR development tools

- Windows 95/98/ME
- Pentium-133 or higher
- At least 4 Mbytes of RAM
- CD-ROM Drive

AVR ISP installation:

Run setup.exe in the CD's avr_isp folder.

Microrobot AVR GCC installation

Refer to the 'Microrobot AVR GCC User Guide.pdf' file in the CD's MaroGcc0.9C folder.

2. How to use Microrobot AVR GCC

Refer to the 'Microrobot AVR GCC User Guide.pdf' file in the CD's MaroGcc0.9C folder.

3. How to use AVR ISP(In-System Programming)

Refer to the 'AVR ISP Manual for AVR 4433 Board.pdf' file.

PART 4 : How to Operate

1. Compile and Download

Compile the source file and download the executable file in the following order.

- Put four batteries into the battery holder and insert the power connector to J1 of the Main PCB.
- Connect the downloading adapter to the PC printer port. Then connect the downloading adapter and the CPU board by using the ribbon cable.
- Turn on the power switch S1 on the control board. LED D1 turns on.
- Copy the Source folder on the CD to the C:\Source. Remove (uncheck) read-only file attributes for the all copied files.
- Run the Microrobot AVR GCC.
- Open c:\source\main.c.
- Select 'Build → Build Option …→ General tab'. The Build Option window appears.
  Select 'Intel hex' as a Hex format, 'at90s4433' as a microcontroller. Check 'Object file' and 'Rom file' boxes in the Generation menu. Type c:\source as an Output Directory and click on OK.
- Select 'Build → Build Option …→ Compiler tab'. Click on the 'Default' button and select 'Size' radio button in the Optimization box. Do not use 'Speed' optimization.
- Select 'Build → Build Option …→ Linker tab'. Click on the 'User Link script' radio button and select 'C:\Program Files\Microrobot\Microrobot AVR GCC\Avr\lib\ldscripts\avr4433.x' and click on OK.
- Press F7 or select 'Build → Build' menu to build.
  (main.c editing window must be selected before building it if there are more than two source editing windows.)
- If you see the following message: 'warning: asm operand1 probably doesn't match constraints' in the output message window, press F7 again.
- 'Create ROM file. Build complete!' message appears in the output message window.
- Run the Atmel AVR ISP.
- Select 'Project → New Project' menu. Select AT90S/LS4433.
- Click on the Program Memory window. Select 'File → Load' menu and open c:\source\main.rom.
- Select 'Project → Save Project' menu and save the project as c:\source\main.avr.
- Select 'Options → Advanced...', Check 'ð Disable Signature Check' box in the Advanced Options message box.

- Select ‘ Program → Auto-Program Options’ menu. Check boxes properly.
- Press F5 for the ‘ Auto Programming’ .
- Take a look at the downloading error messages in the output message window. If there is no ‘ verify error’ , uncheck the ‘ Verify Device check’ box in Auto-Program Options.
- Remove the ribbon cable from the control board and restart the board.

2. R/C servomotor(HES-288)

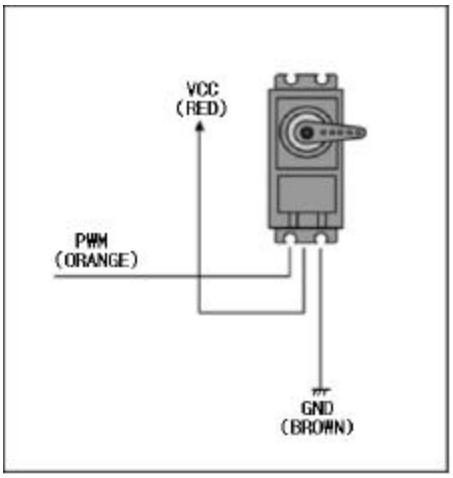
An R/C servomotor is a special type of geared motor. It has its own internal controller, which uses feedback to control the DC motor according to the PWM (Pulse Width Modulation) position signal. The degree of turn of the servomotor's axle depends upon the input pulse width signal. Generally, it turns from 0° to 180°(or -90°~+90°). Therefore the servomotor does not need an encoder to get the axle' s current position.

HES-288 is an inexpensive general servomotor. It is controlled using three wires, as is general servomotor. Two wires are for the power input and the other wire is for the PWM control signal input.

HES-288 Specification

- Torque : 2.5 kg/cm
- Speed : 60 degrees/0.2sec
- Size : 41 x 20 x 35 mm (length x depth x height)
- Weight : 42 g
- Voltage : 4 ~ 6 V
- Operating Frequency : 50Hz~100Hz (Period: 10msec~20msec)

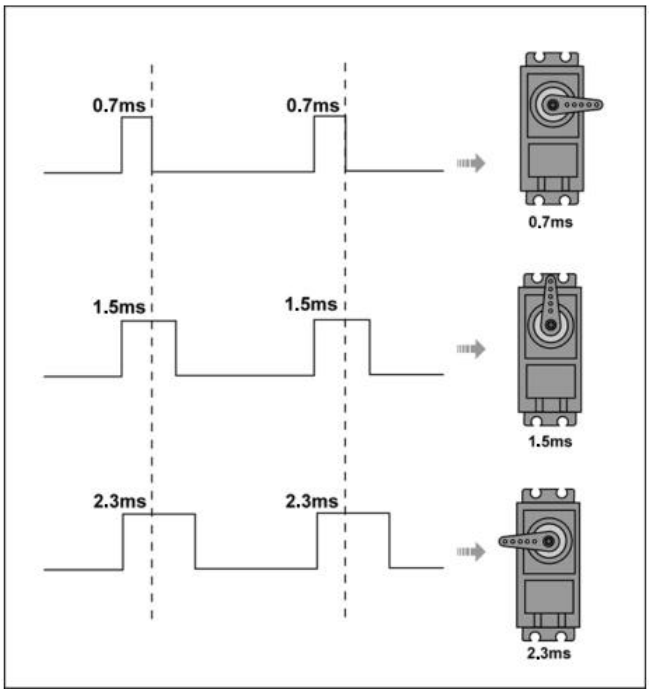
Wire connection



Red: Vcc(DC 4~6V)
Brown: Gnd
Orange: PWM control signal input.

Fig 4.1

Control signal



1.5 msec : 0°(Center)

0.7 msec : + 90°

2.3 msec : - 90°

To keep the desired position (or holding torque), the pulses must be provided every from 10 ms to 20 ms.

Fig 4.2

3. Adjusting servomotors

When the power is on, the servos, which are connected to the board, should turn to the center but they might not. This is because the characteristics of the servomotors are a little different. To adjust each motor, the initial setting values in source code must be changed.

- The AT90S4433 CPU board can control eight servomotors at the same time. The eight servomotors are named SERVO_0, SERVO_1,and SERVO_7 in the program source.
- On the upper part of the PCB, index numbers from 1 to 16 are marked for the servomotor ports. Refer to Fig 1.1 above. Port numbers 1 to 8 correspond to SERVO_0 to SERVO_7 respectively.
- Run MICROROBOT AVR GCC and open the file main.c.
- At the beginning of the source code, there are some user-definitions.

  /****************** User definition ***************************/

  ```
  #define SERVO_0      (1500*8)
  #define SERVO_1      (1550*8)
  #define SERVO_2      (1450*8)
  #define SERVO_3      (1400*8)
  #define SERVO_4      (1350*8)
  #define SERVO_5      (1550*8)
  #define SERVO_6      (1480*8)
  #define SERVO_7      (1500*8)
  ```

  Note : The actual values might differ from those above.

  The defined values are initial center (pulse) values for each servomotor. The center value refers to a time constant which makes a servomotor' s axle turned to the center (0°). Refer to Fig 4.2. The user must determine these center values with actual servomotor test. In the above definition, (1500*8) means 1.5 ms, which is a typical center value. When the value is changed by ±1*8, the axle turns approximately ±0.1125°. Repeat source changing, compiling and downloading to determine the final correct center values.

  For example, when one servomotor' s axle turns 5 degrees count clockwise past the center position, subtract 44(=5/0.1125) from the initial value.

- Below are some other definitions in the source file.

  ```
  #define FACTOR_0     1024
  #define FACTOR_1     1024
  #define FACTOR_2     950
  #define FACTOR_3     1024
  ```

```
#define FACTOR_4        1023
#define FACTOR_5        1024
#define FACTOR_6        930
#define FACTOR_7        1024
```

Note: The actual values might differ from the above.

The above values (constants) are used as turning factors. To make the servos turned properly, the above values should be adjusted after testing.

There is a special test mode to determine the above constants. Press and release the S3(Reset) key while pressing the S2 key and then release the S2 key. Each servomotor should turn 90°. If not, change the corresponding values. For example, if the SERVO_0 turns more than 90 degrees, reduce FACTOR_0. If the SERVO_2 turns less than 90 degrees, increase FACTOR_2.

- After determining the proper values, push the S3 (Reset) key for the reset and push S2 key to start the user routine.

PART 5 : Source Codes

Source files' description.

- main.c : Main program.
- io.h : Header file which has AVR CPU' sregisters and I/O pins definition.
- signal.h : Interrupt vectors functions and interrupts vector table definition.
- init.c : Basic initialization.
- function.c : Delay, buzzer and switch functions.
- motion.c : Servomotor control functions.
- inter.c : Interrupt routine functions.

1. Init.c

```
void port_init(void)
{
      outp( 0xfe, DDRB );     //LSB of the port B = input pin, the other pins = output pin.
      outp( 0xff, DDRC );     //Set all PORTC pins to output.
      outp( 0xff, DDRD );     //Set all PORTD pins to output.

      outp( 1, PORTB );       //Set the bit0 of the Port B to use the internal full-up resistor.
      outp( 0, PORTC );
      outp( 0, PORTD );
}
```

Initializes ports.

```
void motor_init(void)
{
      outp(4, TCCR0);                 // TIMER_0 CK/256 -> 1=32usec

      outp(0, TCCR1A);
      outp(1, TCCR1B);                // CTC1 clear, TIMER_1 CK/1 -> 8=1usec

      __outw(0xffff, OCR1L);          // 65535 -> (65535/8)usec = 8.192msec
}
```

Initializes two timers to generate continuous pulse with constant period (20 ms), which are used to control the servomotors.

MCU main clock is 8 MHz (1/8 us). The timer0' s prescaler is set to 256. It means that the timer0 clock is 32 us (1/8 us * 256). The timer1 uses the main clock (1/8 us) without prescaling.

```
void motor_enable(void)
{
      outp(178,TCNT0);              // 256-178=78 -> 78*32usec=2.496msec
      __outw(0, TCNT1L);

      cbi(TIFR, TOV0);              // clear OCIF1 : Output Compare Flag1 clear
      sbi(TIMSK, OCIE1);            // set OCIE1 : Timer1 Output Compare Interrupt enable

      cbi(TIFR, OCF1);              // clear TOV0 : Timer0 overflow Flag clear
      sbi(TIMSK, TOIE0);            // set TOIE0 : Timer0 overflow Interrupt enable
}
```

Sets time-constant to the timer0 and enables timer0 interrupt. Even if the timer0 interrupt is enabled, the timer0 interrupt will not occur until the global interrupt bit is enabled.

```
void motor_disable(void)
{
      __outw(0, TCNT1L);
      outp(0,TCNT0);
      outp( 0, PORTD );
      cbi(TIMSK, OCIE1);            // clear OCIE1 : Timer1 Output Compare Interrupt disable
      cbi(TIMSK, TOIE0);            // clear TOIE0 : Timer0 overflow Interrupt disable
}
```

Function to disable the servomotors.

```
void motor_zero(void)
{
	int i;

	Init(0)=SERVO_0;
	Init(1)=SERVO_1;
	Init(2)=SERVO_2;
	Init(3)=SERVO_3;
	Init(4)=SERVO_4;
	Init(5)=SERVO_5;
	Init(6)=SERVO_6;
	Init(7)=SERVO_7;

	Factor(0)=FACTOR_0;
	Factor(1)=FACTOR_1;
	Factor(2)=FACTOR_2;
	Factor(3)=FACTOR_3;
	Factor(4)=FACTOR_4;
	Factor(5)=FACTOR_5;
	Factor(6)=FACTOR_6;
	Factor(7)=FACTOR_7;

	for(i=0; i<8; i++)
	{
		Deg(i)=Speed(i)=0;
		Pulse(i)=Init(i);
	}
}
```

Copies user adjusted (defined) constants to the variables. The constants can be used directly instead of being copied to the variables. However, for the upgrade version the variables are used. In the upgrade version, the variables may be also filled out with constants from the PC.

```
void system_init(void)
{
	port_init();
	motor_init();
	motor_zero();
}
```

System initialization.

2. Function.c

```
void delay(int ticks)
{
	while(ticks--);
}

// 1msec UNIT delay function
void delay_1ms(unsigned int i)
{
	word j;
	while(i--)
	{
		j=2000;   // 8Mhz
		while(j--);
	}
}
```

1 ms based delay function. For example, to make 1 sec time delay, call delay_1ms(1000). This function does not consider other interrupts. It means that other interrupts during the delay function can make the delay time longer.

```
void buzzer(void)
{
	word i;
	for(i=0; i<50; i++)
	{
		BUZZER_ON;
		delay(BUZ_DLY1);
		BUZZER_OFF;
		delay(BUZ_DLY1);
	}

	for(i=0; i<50; i++)
	{
		BUZZER_ON;
		delay(BUZ_DLY0);
		BUZZER_OFF;
		delay(BUZ_DLY0);
	}
}
```

Makes buzzing sound using the BTG-47 buzzer. The BTG-47 is frequency (pulses) activated.

```
void until_switch_intput(void)
{
	loop_until_bit_is_clear(PINB, 0);
	loop_until_bit_is_set(PINB, 0);
}
```

Push button switch S2 input function, which waits until the S2 key is pressed and released.

3. Inter.c

In the source, two different interrupts are used. The interrupts are exclusive. It means, when one interrupt service routine is served, the other one must wait.

```
SIGNAL(_overflow0_)
{
	__outw(0, TCNT1L);
	outp(178,TCNT0);		// 256-178=78 -> 78*32usec=2.496msec

	sbi(PORTD, No);

	if( Pulse(No) > (Init(No)+Deg(No)*71+(Speed(No)/2)) )		Pulse(No)-=Speed(No);
	else if( Pulse(No) < (Init(No)+Deg(No)*71-(Speed(No)/2)) )	Pulse(No)+=Speed(No);
	else  Flag_motor&=~(1<<No);

	__outw(Pulse(No), OCR1L);
	sbi(TIMSK, OCIE1);		// set OCIE1 : Timer1 Output Compare Interrupt enable

	No++;
	No&=0x07;

	cbi(TIFR, TOV0);		// clear OCIE1 : Output Compare Flag1 clear
}

SIGNAL(_output_compare1a_)
{
	outp(0, PORTD);			//Clear PortD.
	cbi(TIMSK, OCIE1);		// clear OCIE1 : Timer1 Output Compare Interrupt disable
}
```

Timer0 overflow interrupt occurs every 2.5 ms. In the interrupt service routine, one of eight-servomotor control pins is activated by turns. It means each servomotor has a controlled-focus every 20 ms(2.5msec*8).

'No' is an index variable that has a port number to be serviced. 'No' can be form 0 to 7. The corresponding port pin is set (high) and the high-pulse time variable Pulse (No) is set to the OCR1L. After the set time elapsed, output_compare1a interrupt occurs, which makes the pulse low.

Speed (No) value is used to control the servomotor turning speed. When the Speed(No) is 71, the servo turns 1°/20ms(50°/sec).

It takes time for a servomotor to turn to its destination. Flag_motor is used to check whether the servomotors are finish tuning. Flag_motor is a one-byte valuable. Each bit stands for each servomotor' s status. 1 means the motor is still turning and 0 means the turning is done.

The servomotor turns 1° when the pulse width is changed by 8.888 us. The timer1' s dock is 1/8 us. Thus the servomotor turns 1° every 71 timer1' sclock ticks.

Init(No) is the time constant in order that the servomotor' saxle gets the center(0°) position.

4. Motion.c

```
void wait_until_move(void)
{
	Flag_motor=0xff;		// Initialze with 0xff( all servos are still turning.)
	while(Flag_motor);		// Wait until all servomotor finish turning.
}
```

Waits until all servomotors' have finished turning before starting a new next action.

```
// Turn the servo to the absolute angle at the speed.
void absolute_deg(int no, int deg, int speed)
{
	Deg(no)=(int)((long)(deg)*(long)(Factor(no))>>10);
	Speed(no)=(int)((long)(speed)*(long)(Factor(no))>>10);
}
```

Turns the servo(no) to the absolute angle(deg) at the given speed(speed).

Shift operation(>>) is used instead of divide operation to save the operation speed.

The Factor(no) is used to calculate the Speed(no) as well as the Deg(no). It is because each motor has a different characteristic.

```
void same_absolute_deg(byte flag, byte dir, int deg, int speed)
{
	int i;

	wait_until_move();
	for(i=0; i<8; i++)
	{
		if( flag&(1<<i) )
		{
			if(dir&(1<<i))  absolute_deg(i, deg, speed);
			else 			absolute_deg(i, -deg, speed);
		}
	}
}
```

Moves the motors(up to eight) at the same time to the same absolute angle. The angle(deg) and the speed are applied to all enabled servomotors equally, but the direction can be selected with the dir parameter : clockwise or counterclockwise.

<Parameters>

flag : Motor Enable flags. Each bit is a corresponding servomotor' s enable. 1 = enable, 0 = disable.

dir : Direction flags for the each motor. 1= use deg, 0 = use -deg.

deg : Absolute angle to turn.

speed : turning speed.

This function is used when several motors need to move to the same angle or symmetric angle.

For examples,

same_absolute_deg(S0|S2|S4|S6, S2|S4, 45, speed);

-Enables the motor 0,2,4,6.

-Motor 1,3 turns to the –45 degree position and motor 0,4 turns to the 45 degree position with the speed.

same_absolute_deg(S0|S2, S2, 15, speed);

-Enables the motor 0,2.

-Motor 0 turns to the –15 degree position and motor 2 turns to the 15 degree position with the speed.

```
// relative angle turning function for the each motor.
void relative_deg(int no, int deg, int speed)
{
	int temp;
	temp=(int)(((long)(Deg(no))<<10)/(long)(Factor(no)));
	deg+=temp;
	Deg(no)=(int)((long)(deg)*(long)(Factor(no))>>10);
	Speed(no)=(int)((long)(speed)*(long)(Factor(no))>>10);
}
```

Turns motors by the relative angle at the given speed.

```
void same_relative_deg(byte flag, byte dir, int deg, int speed)
{
	int i;

	wait_until_move();
	for(i=0; i<8; i++)
	{
		if(flag&(1<<i))
		{
			if(dir&(1<<i))  relative_deg(i, deg, speed);
			else			relative_deg(i, -deg, speed);
		}
	}
}
```

This function can be used when several motors need to move by the same relative angle or -angle. This is a same function as the same_absolute_deg() except that the deg is a relative angle.

5. Main.c

```
typedef unsigned char byte;
typedef unsigned int word;

#define sei()  asm volatile ("sei" ::)
#define cli()  asm volatile ("cli" ::)
```

#define sei() asm volatile ("sei" ::)

Inline-assembler definition for the convenience. sei() changes to sei(assembler instruction).

```
void adjust_mode(void)
{
	until_switch_intput();
	same_absolute_deg(S0|S1|S2|S3|S4|S5|S6|S7, 0, 90, 100);
	same_absolute_deg(S0|S2, S2, 90, 100);
	buzzer();                                   //Buzzing sound
}
```

Special test mode function to adjust all servomotors. Press and release the S3 (Reset) key while pressing the S2 key and then release the S2 key to enter this mode. Each servomotor turns 90°.

```
int main(void)
{
        int i;

        system_init();                  // System Initialize
        buzzer();                       // Buzzer
        motor_enable();

        sei();                          // Global interrupt enable

        if(bit_is_clear(PINB,0))    adjust_mode();

        until_switch_intput();
        delay_1ms(500);

        buzzer();
        delay_1ms(2000);

        while(1)
        {
                // User own functions
        }
}
```

Main function: Initialization, interrupt enable and user functions(blank).